Elaboración de un web scraper.

Introducción.

Las extensiones y librerías agregadas en Python permiten al programador o desarrolladores a elaborar y estructurar sistemas cuyas funciones satisfagan de manera eficaz y eficiente las necesidades planteadas por el equipo desarrollador o condiciones impuestas por un cliente empleador. Una de las herramientas ejemplificadoras de estas es a la que corresponde el presente documento para su elaboración: Un web scraping.

Un web scraping es una herramienta utilizada para la identificación, recolección, y almacenamiento de información o data obtenida de páginas web que resulten de interés para el cliente o desarrollador. Una herramienta que permite la exhibición de la información obtenida en data frameworks organizados para su fácil consulta y exportación.

A continuación, se presenta una guía detallada de la fundamentación, elaboración y utilización de esta herramienta, incluyendo a su vez recomendaciones y observaciones para su fácil desarrollo.

Pasos preliminares: Extensiones y librerías a utilizar.

La obtención de información de la web y su exhibición organizada de esta son funciones correspondientes a las dos principales librerías utilizadas en este proyecto: *Selenium* y *pandaSQL*.

Selenium es un software de código abierto utilizado para la validación de aplicaciones web en distintos navegadores y plataformas. Por otra parte, *pandaSQL* es una librería de Python que permite la manipulación de un *Dataframe* de Pandas usando SQL, creándose así una tabla SQLite a partir del marco de datos de interés de Pandas que permite a los usuarios su consulta mediante SQL.

Pasos preliminares: Instalaciones necesarias.

Posteriormente al inicializar el software de preferencia utilizado para la programación de Python, es necesario abrir la terminal incluida de esta donde se introducirán los comandos correspondientes a cada librería.

En el caso de *Selenium*, se tiene que el comando utilizado para su instalación es:

```
pip install --user selenium
```

Por otra parte, para *pandaSQL*, el comando a introducir es:

```
pip install pandasql
```

Observación: Con el propósito de evitar problemas de importaciones faltantes o localización de las librerías, es recomendable agregar el siguiente argumento para la especificación exacta de la locación en donde se instalarán las librerías.

```
/full/path/to/python -m pip
```

Para permitir que *Selenium* pueda manipular y controlar las interacciones con los sitios web visitados, es necesaria la instalación de un driver correspondiente al navegador web a utilizar y específico a la versión de tal navegador. Para el caso de este proyecto el navegador utilizado fue Chrome, por lo que se proporciona una liga con las descargas posibles del driver de acuerdo a la versión del navegador.

sites.google.com/chromium.org/driver/

Pasos preliminares: Importación de librerías a utilizar.

Las librerías a utilizar también incluirán algunas preinstaladas en Python para la modificación y presentación del formato y texto, obtención de la fecha actual, métodos de encadenado, comandos de espera basado en tiempo, entre otros. En consecuencia, las siguientes importaciones son necesarias para el correcto funcionamiento del código.

```
from urllib.request import urlopen
import urllib.request
import requests
import time
from multiprocessing import Process, Queue, Pool
import threading
import sys
import re
from datetime import datetime
```

La importación de *pandas*, *pandaSQL*, y *numpy* también serán realizadas. Es recomendable asignar un alias a aquellas librerías cuyas funciones se usen más como es el ejemplo de *numpy* a *np* en la siguiente entrada de código.

```
import pandas as pd
import pandasql as ps
import numpy as np
```

Finalmente, se realizará la importación de *Selenium* y algunas de sus funciones: *webdriver* que permitirá el control remoto de las páginas web; *keys* son palabras clave para la implementación de ciertas acciones; *by* que solicita instancias basados en tipos de carácter, métodos o propiedades; *expected_conditions*, una extensión que contiene posibles situaciones y condiciones de las páginas web analizadas por default; *webDriverWait* condiciones que pausan el proceso del programa para su correcto funcionamiento ;y *Options*, clase que permite la optimización del código en el navegador web.

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
```

```
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
```

Driver y Path.

Posterior a la correcta importación de las librerías a utilizar, es necesario establecer la conexión entre el driver controlador del navegador web con *Selenium*. Dos variables son definidas con nombres "path" y "driver", denominaciones sugeridas a replicar para su fácil identificación.

La definición de la variable "path" es sólo para el almacenamiento de la localidad del archivo ejecutable chromedriver descargado con anterioridad. Esta variable deberá ser igualada a tal extensión.

```
path = r"C:\Users\---\chromedriver.exe"
```

Por otra parte, driver será definido como webdriver.Chrome, entre paréntesis se colocará la variable path.

```
driver = webdriver.Chrome(path)
```

Así, Selenium tendrá conexión directa con el driver para el control y manipulación remota del navegador web.

Observación: Si el proyecto se pretende compartir con otros compañeros desarrolladores o si existen complicaciones respecto a la instalación del driver y la incapacidad de Python para poder detectarlo, es recomendable la importación del asistente automático del driver. Así, el programa automáticamente descargará e instalará en Python el driver correspondiente al navegador web de cada integrante sin necesidad de tener que modificar la ubicación del archivo ejecutable por separado en cada equipo. En consecuencia, al ya ser necesaria la especificación de la locación del archivo, la variable "path" no es utilizada y la variable "driver" es directamente igualada al instalador automático del driver.

```
from webdriver_manager.chrome import ChromeDriverManager
driver = webdriver.Chrome(ChromeDriverManager().install())
```

Primera interacción.

Una vez establecidas las librerías utilizar y la correcta conexión con el controlador web, se puede proceder a realizar la acción más simple de Selenium: El abrir una página web. Es claro que esta página web debe ser de interés para los desarrolladores o empleador, ya que será de esta de donde se obtengan los datos e información requeridos para el proyecto.

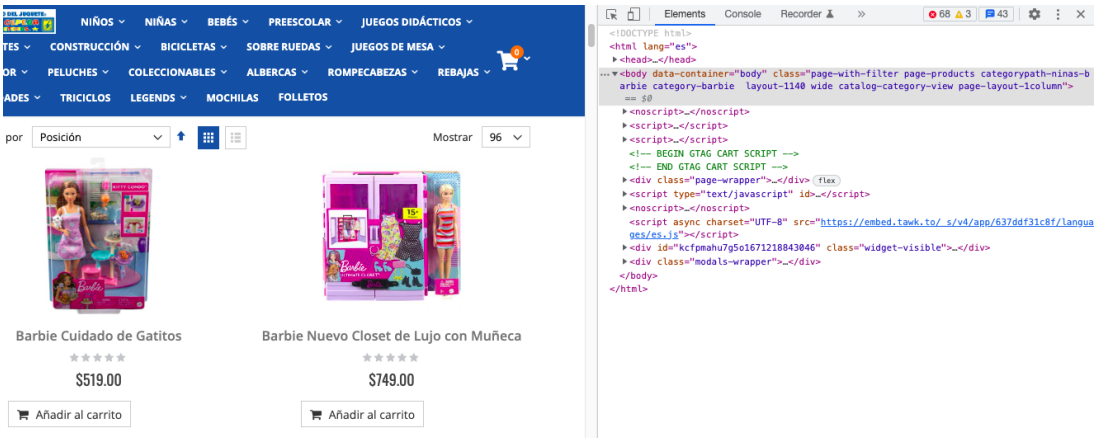
A continuación, se establece una variable almacenadora del url correspondiente a la página a examinar. El primer comando a utilizar con selenium es driver.get cuyo parámetro a ingresar es la url de la página a examinar. Posteriormente, el programa desplegará una nueva ventana en el navegador web de la página ingresada. La implementación de time.sleep es para asegurar la correcta carga completa de la página web antes de proceder con cualquier otra acción, esto con el objetivo de no generar errores de carga o de acciones ya que Selenium podría proceder a ejecutar las siguientes acciones del código antes de que la página se encuentre completamente cargada.

```
url = "url_de_pagina_web"
driver.get(url)
time.sleep(10)
```

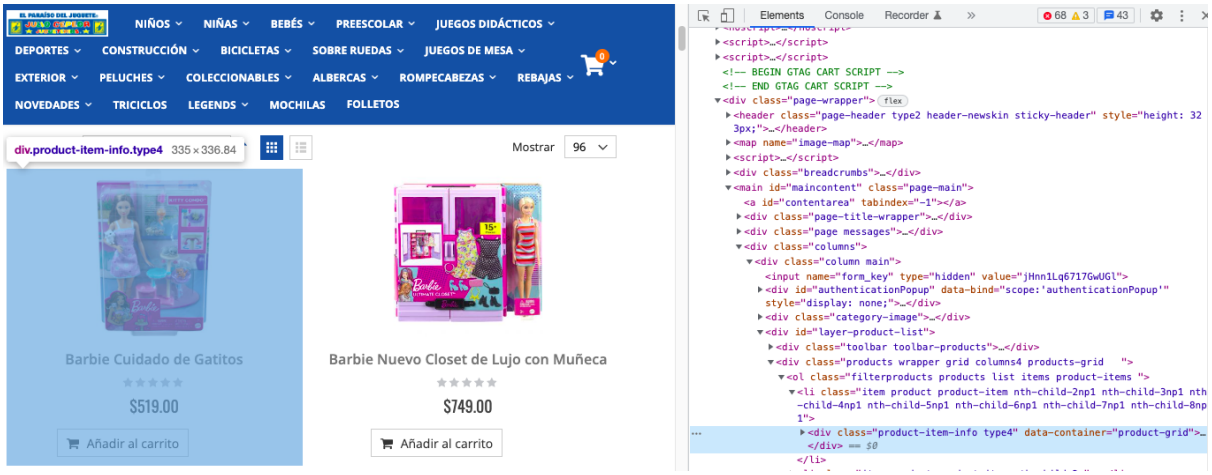
Para una mejor ejemplificación de los procesos subsecuentes, se hará uso de web scraper con temática de jugueterías que adquiere y almacena nombres, marcas, precios, y precios con descuento.
Realizando el correspondiente seguimiento a las instrucciones posteriores, se introduce el url de la página web de interés y se establece una variable que lo almacena. Nótese que para este ejemplo en particular se agregó el parámetro "+juguete" esto con la intención de poder cambiar el juguete de interés a obtener información.

```
url_1="https://juliocepeda.com/catalogsearch/result/?q="+juguete
driver.get(url_1)
time.sleep(10)
```

Obtención de datos.
A continuación será necesario el acceder a la página web de interés para obtener los parámetros necesarios de la información de interés a recolectar. Para el ejemplo de la juguetería, se toman en cuenta las siguientes condiciones: Juguete, autoservicio, nombre, marca, precio, y precio de promoción.
Acceda entonces a la página de interés y despliegue las herramientas de desarrollador.



Para la inicialización del dataframe el primer dato a considerar será el nombre del producto, por lo tanto, es necesario desplegar las divisiones mostradas en las herramientas de desarrollador hasta encontrar aquella sección que almacena la información del nombre.



Nótese que la división almacenadora del dato sombrea en la página web el dato de interés buscado, esto permite que la obtención de las divisiones contenedoras de información sea sencilla e intuitiva.

Una vez identificada la división de interés es necesario determinar el tipo de elemento que es, en este ejemplo en particular, se puede observar que la división que contiene el nombre del juguete correspondiente es un elemento tipo "class" nombrado como "product-item-into.type4".

A continuación, de vuelta al código, se establecerá una variable denominada "juguetes" que guarde la información obtenida de la página web implementando la función "find_elements" y utilizando la información proporcionada por las herramientas de desarrollador.

```
juguetes=driver.find_elements_by_class_name("product-item-info.type
4")
```

Nótese que en la sección "by_class_name" se utilizó tal parámetro ya que ese es el tipo de elemento que almacena el nombre del producto. Posterior a esto, se introdujo el nombre de tal variable que guarda el nombre del producto.

Listas almacenadoras.

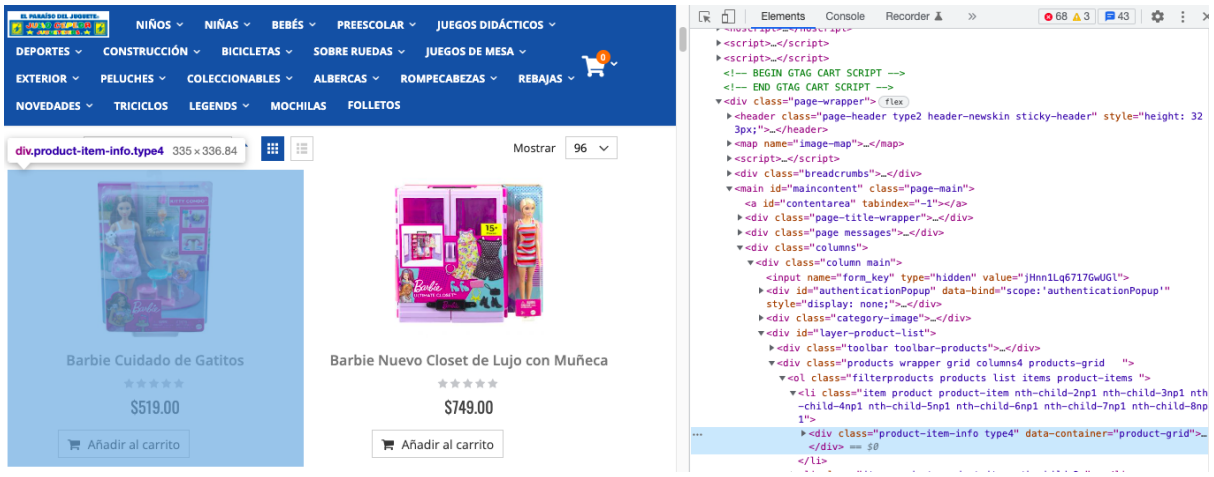
La información inicial ha sido obtenida, una clave única numerada sobre el conjunto de números naturales ha sido creada. A continuación, es necesaria la creación de listas que almacenen los datos de interés de los juguetes. Defínase entonces una lista vacía denominada con el nombre del dato que se almacenará en esta. Para el ejemplo de los juguetes, se continuará utilizando el dato del nombre.

A continuación se recorren los índices obtenidos de la primera lista creada que contiene la clave única de cada producto. Sobre esta se implementará un código "try-except". Durante el proceso de obtención de información, puede presentarse el evento de que alguno de los

productos no contengan la información buscada del producto, en tal caso, se opta por sustituir la falta de datos por un valor default “nan” que indica tal caso. Así, el código final resulta de la siguiente manera.

```
lista_nombres = []
for i in range(0,len(juguetes)):
   try:
        lista_nombres.append(juguetes[i].text)
   except:
        lista_nombres.append(np.nan)
```

Si bien la información de cada producto en la página web de interés está contenida en ciertas divisiones visibles en las herramientas de desarrollador, es necesario el acceder a cada una de las páginas individuales de los productos mostrados en esta para la correcta obtención de los datos buscados. El procedimiento de esta acción es análoga al proceso realizado hasta el momento con la página principal de interés, sin embargo, para la automatización de este, se hará uso de un ciclo for que ejecute el proceso para cada uno de los productos desplegados.
A continuación, es necesaria la creación de una lista cuyo contenido sean los urls de los productos individuales, esto con el propósito de acceder a cada uno de ellos y de poder iterar al momento de establecerse un ciclo “for”.
Haciendo uso de las herramientas de desarrollador se puede ubicar la división que almacena los urls de cada uno de los productos individuales desplegados.



Una vez identificados, se crea la respectiva lista vacía que los almacenará.

```
list_urls = []
juguetes=driver.find_elements_by_class_name("product-item-info.type
4")
time.sleep(5)
```

```
for i in range(0,len(juguetes)):
   try:
list_urls.append(juguetes[i].find_element_by_tag_name("a").get_attr
ibute("href"))
   except:
        list_urls.append(np.nan)
time.sleep(5)
```

Similarmente al ejemplo de obtención de nombres, cabe la posibilidad de que un url no este definido para algún producto, por ello, se procede a depurar la lista de urls sustituyendo aquellas excepciones que no poseen urls con el valor default “nan”.

```
list_urls = [i for i in lista_urls if str(i) != "nan"]
```

Teniendo la anterior lista completada, la iteración para obtener la información de los productos individuales puede ser realizada.
Para nuestro ejemplo en particular, se estableció la recolección de los datos de marca, precio, y precio de promoción. Por lo tanto, será necesaria la correspondiente creación de listas almacenadoras de cada una.
El ciclo “for” estará establecido respecto a la cantidad de urls obtenidos, y la obtención de los datos seguirá un procedimiento análogo al utilizado con anterioridad: Ubicar la división que contiene el dato con ayuda de las herramientas de desarrollador, identificar que tipo de elemento lo contiene, establecer un código “try-except” para generar valores default “nan” en caso de que un dato no sea encontrado, y almacenar la información en la lista correspondiente. Así, se obtiene:

```
list_marca=[]
list_precio=[]
list_precio_promocion=[]
list_urlim = []
for j in list_urls:
   driver.get(j)
   time.sleep(8)
   try:
               list_marca.append(driver.find_elements(By.CLASS_NAME,
"base")[0].text)
   except:
        list_marca.append(np.nan)
```

```
    try:
            lista_precio.append(driver.find_elements(By.CLASS_NAME,
"price-wrapper
")[0].text.replace("$","").replace("'","").replace("'",""))
    except:
        list_precio.append(np.nan)
    try:

lista_precio_promocion.append(driver.find_elements(By.CLASS_NAME,
"old-price")[0].text.replace("$","").replace("'","").replace("'",""
))

    except:
        list_precio_promocion.append(np.nan)

    try:

list_urlim.append(juguetes[i].find_element_by_tag_name("img").get_a
ttribute("data-src"))
    except:
        list_urlim.append(np.nan)
```

El código anterior, recorre cada uno de los urls de cada producto individual, los abre, obtiene la información solicitada y la almacena hasta que el último url haya pasado por este proceso.

La obtención de la fecha de consulta de los productos individuales puede ser obtenida mediante la librería incluida en Python “datetime”. Así, con un sencillo formato de día, mes, y año, la fecha puede ser incluida en una correspondiente lista vacía especializada en el almacenamiento de este dato.

```
ahora = datetime.now()
fecha = ahora.strftime("%d/%m/%Y")
list_fecha = []
for i in juguetes:
    list_fecha.append(fecha)
print(len(list_fecha))
```

Formato y presentación.

La información ha sido almacenada, ahora es necesaria la exhibición de esta mediante las librerías de *pandas*.

Utilizando la función "*DataFrame*" se establecerán los campos o columnas a exhibir en el data frame. La fecha de consulta, el juguete en específico del que se busco la información, el establecimiento de autoservicio, el nombre, imágen de referencia, marca, precio, y precio de promoción. Son los datos de interés para el ejemplo específico de los juguetes.

Se establece entonces lo planteado con anterior con la siguiente línea de código:

```
df_jugueteria=pd.DataFrame(columns=["FECHA","JUGUETE",
"AUTOSERVICIO","NOMBRE","PRECIO","PRECIO_PROMOCION"])
```

Donde la variable df_jugueteria es creada para la simplificación de los procesos siguientes.

A continuación, cada uno de los campos es completado con las listas creadas almacenadoras de la información correspondiente.

```
df_jugueteria["FECHA"] = list_fecha
df_jugueteria["AUTOSERVICIO"] = "Julio Cepeda Jugueteria"
df_jugueteria["NOMBRE"] = list_nombres
df_jugueteria["PRECIO"] = list_precio
df_jugueteria["JUGUETE"] = juguete
df_jugueteria["PRECIO_PROMOCION"] = list_precio_prom
df_jugueteria["IMAGEN"] = list_urlim

df_juguetron["PRECIO"] = lista_precio
df_juguetron["PRECIO_PROMOCION"] = lista_precio_promocion
```

Nótese que, para este ejemplo en particular, el campo de autoservicio fue completado con un string constante "Julio Cepeda Juguetería", esto ya que la obtención de la información solo fue realizada en la página web de esta juguetería.

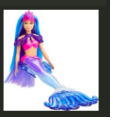
La exhibición de una imágen de referencia de los productos implica la definición de una función sencilla que recolecta la información almacenada en la lista de imágenes para brindarle un formato adecuado para su exhibición.

```
def imagen(path):
   return '<img src="'+ path + '" width="100" >'
display(HTML(df_jugueteria.to_html(escape=False,formatters=dict(IM
AGEN=imagen))))
```

Finalmente, un filtro aplicado al formato es utilizado. Para el caso específico de los nombres de los productos, estos serán convertidos a un tipo de dato string; su contenido será modificado para que el texto sea desplegado en su totalidad en mayúsculas; y en caso de que un producto no requerido haya logrado ser almacenado hasta al momento, se aplicará un filtro al texto para que se eliminen estos productos infiltrados. En el ejemplo específico de la juguetería, el producto a buscar fueron barbies, por lo tanto, el filtro aplicado fue el que la cadena de texto contenga la palabra "BARBIE".

```
df_juguetron.NOMBRE = df_juguetron.NOMBRE.astype(str)
df_juguetron.NOMBRE=df_juguetron.NOMBRE.str.upper()
df_juguetron=df_juguetron[df_juguetron.NOMBRE.str.contains("BARBIE")]
```

Así, finalmente, el resultado final es un data frame obtenido de nuestro web scrapping desarrollado en este documento.

	FECHA	JUGUETE	AUTOSERVICIO	NOMBRE	PRECIO	PRECIO_PROMOCION	IMAGEN
0	14/12/2022	Barbie	Julio Cepeda Jugueteria	Barbie Cuidado de Gatitos	519.00	519.00	
1	14/12/2022	Barbie	Julio Cepeda Jugueteria	Barbie Nuevo Closet de Lujo con Muñeca	749.00	749.00	
2	14/12/2022	Barbie	Julio Cepeda Jugueteria	Barbie Cutie Reveal Venado	649.00	649.00	
3	14/12/2022	Barbie	Julio Cepeda Jugueteria	F P Little People Barbie Casa de los Sueños	1,799.00	1,799.00	
4	14/12/2022	Barbie	Julio Cepeda Jugueteria	Barbie Mermaid Power Sirena Malibu	519.00	519.00	

Conclusiones.

Después de la realización de este web scrapping, se podrá notar como realmente es una herramienta útil para la recolección de datos de una página web, y junto con la ayuda de los dataframe se logra una correcta organización y visualización de los datos. Las herramientas presentadas en este documento no solo son necesarias para la realización de un web scrapping, si no que ayudan a desarrollar más habilidades de programación ya que se implementan diversas librerías propias de Python y a su vez se comienza a incentivar el interés con respecto al cómo están diseñadas las páginas web. De igual manera, la elaboración de este proyecto permite dar un acercamiento a lo que son las bases de datos, logrando entender un poco sobre cómo están estructuradas.